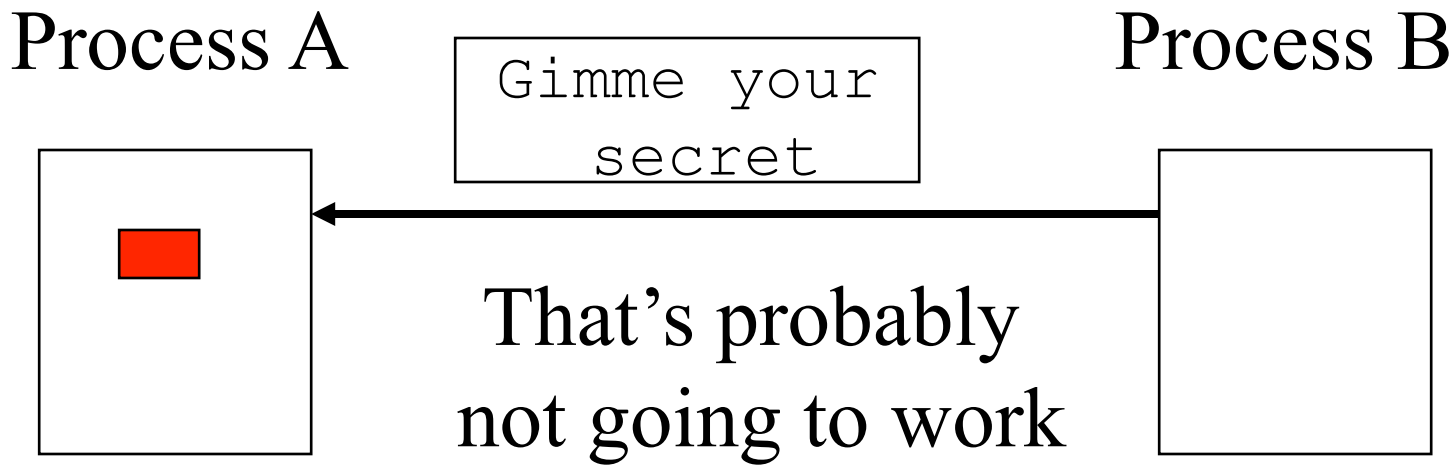
Protecting Interprocess Communications

- Operating systems provide various kinds of interprocess communications
  - Messages
  - Semaphores
  - Shared memory
  - Sockets
- How can we be sure they're used properly?

IPC Protection Issues

- How hard it is depends on what you're worried about
- For the moment, let's say we're worried about one process improperly using IPC to get info from another
  - Process A wants to steal information from process B
- How would process A do that?

Message Security



Can process B use message-based IPC to steal the secret?

How Can B Get the Secret?

- He can convince the system he's A
  - A problem for authentication
- He can break into A's memory
  - That doesn't use message IPC
  - And is handled by page tables
- He can forge a message from someone else to get the secret
  - But OS tags IPC messages with identities
- He can "eavesdrop" on someone else who gets the secret

Can an Attacker Really Eavesdrop on IPC Message?

- On a single machine, what is a message send, really?
- A copy from a process buffer to an OS buffer
  - Then from OS buffer to another process' buffer
  - Sometimes optimizations skip some copies
- If attacker can't get at processes' internal buffers and can't get at OS buffers, he can't "eavesdrop"
- Need to handle page reuse (discussed earlier)
- Also an issue for properly checking authorization (discussed earlier)

Other Forms of IPC

- Semaphores, sockets, shared memory, RPC
- Pretty much all the same
  - Use system calls for access
  - Which belong to some process
  - Which belongs to some principal
  - OS can check principal against access control permissions at syscall time
  - Ultimately, data is held in some type of memory
    - Which shouldn't be improperly accessible

So When Is It Hard?

1. When there's a bug in the OS
   - E.g., not always checking authorization
   - Allowing masquerading, eavesdropping, etc.
   - Or, if the OS itself is compromised, all bets are off
2. What if it's not a single machine?
3. What if the OS has to prevent cooperating processes from sharing information?

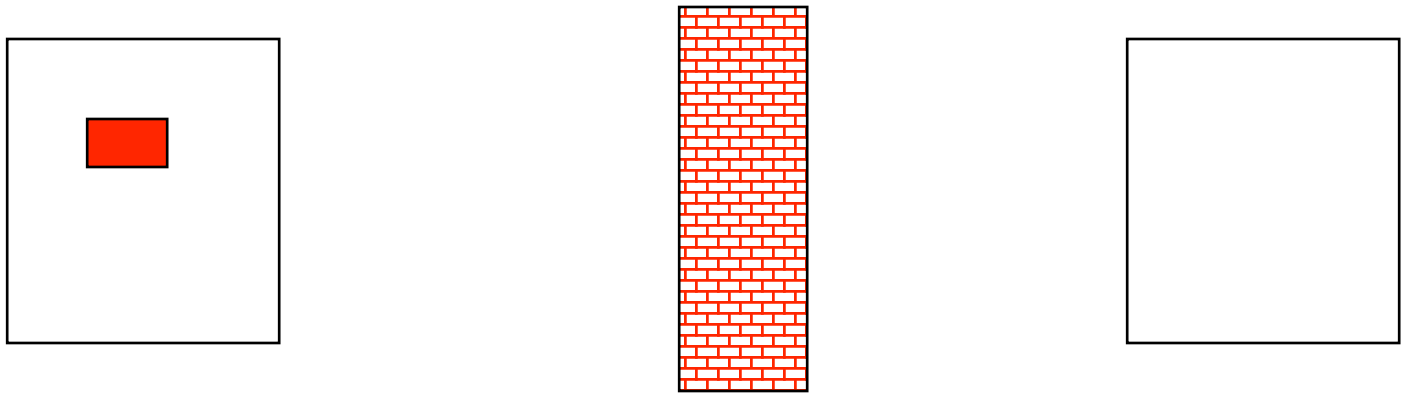
Distributed System Issues

- What if your RPC is really remote?
- RPC tries to make remote access look "just like" local access
- The hard part is authentication
  - The call didn't come from your OS
  - How do you authenticate its origin?
- With usual remote authentication and authorization mechanisms

The Other Hard Case

Process A

Process B



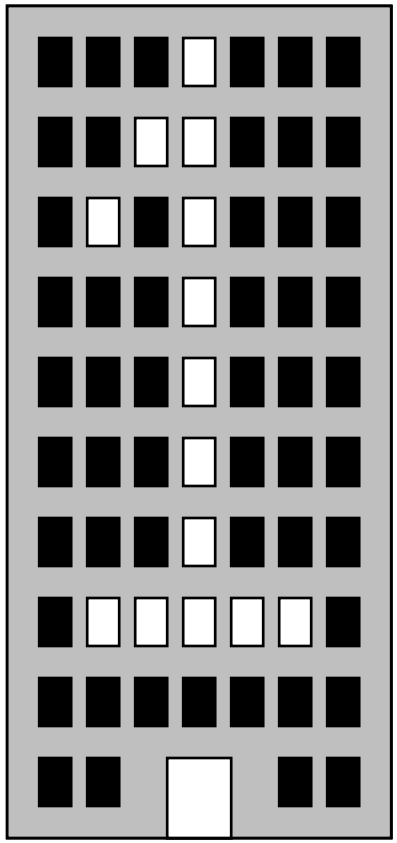
Process A wants to tell the secret to process B
But the OS has been instructed to prevent that
A necessary part of Bell-La Padula, e.g.
Can the OS prevent A and B from colluding
to get the secret to B?

OS Control of Interactions

- OS can "understand" the security policy
- Can maintain labels on files, process, data pages, etc.
- Can regard any IPC or I/O as a possible leak of information
  - To be prohibited if labels don't allow it

Covert Channels

- Tricky ways to pass information
- Requires cooperation of sender and receiver
  - Generally in active attempt to deceive system
- Use something not ordinarily regarded as a communications mechanism



Covert Channels in Computers

- Generally, one process "sends" a covert message to another
  - But could be computer to computer
- How?
  - Disk activity
  - Page swapping
  - Time slice behavior
  - Use of a peripheral device
  - Limited only by imagination

Handling Covert Channels

- Relatively easy if you know details of how the channel is used
  - Put randomness/noise into channel to wash out message
- Hard to impossible if you don't know what the channel is
- Not most people's problem

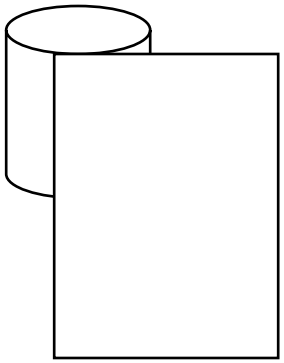
Stored Data Protection

- Files are a common example of a typically shared resource
- If an OS supports multiple users, it needs to address the question of file protection
- Simple read/write access control
- What else do we need to do?
- Protect the raw disk or SSD

Encrypted File Systems

- Data stored on disk is subject to many risks
  - Improper access through OS flaws
  - But also somehow directly accessing the disk
- If the OS protections are bypassed, how can we protect data?
- How about if we store it in encrypted form?

An Example of an Encrypted File System



K_s

Sqamsẹdq
$099 sn
my
sauhmgs
abbntms

Issues for encrypted file systems:

When does the cryptography occur?

Where does the key come from?

What is the granularity of cryptography?

When Does Cryptography Occur?

- Transparently when a user opens a file?
  - In disk drive?
  - In OS?
  - In file system?
- By explicit user command?
  - Or always, implicitly?
- How long is the data decrypted?
- Where does it exist in decrypted form?

Where Does the Key Come From?

- Provided by human user?
- Stored somewhere in file system?
- Stored on a smart card?
- Stored in the disk hardware?
- Stored on another computer?
- Where and for how long do we store the key?

What Is the Granularity of Cryptography?

- An entire disk?
- An entire file system?
- Per file?
- Per block?
- Consider both in terms of:
  - How many keys?
  - When is a crypto operation applied?

What Are You Trying to Protect Against With Crypto File Systems?

- Unauthorized access by improper users?
  - Why not just access control?
- The operating system itself?
  - What protection are you really getting?
  - Unless you're just storing data on the machine
- Data transfers across a network?
  - Why not just encrypt while in transit?
- Someone who accesses the device not using the OS?
  - A realistic threat in your environment?

Full Disk Encryption

- All data on the disk is encrypted
- Data is encrypted/decrypted as it enters/leaves disk
- Primary purpose is to prevent improper access to stolen disks
  - Designed mostly for portable machines (laptops, tablets, etc.)

HW Vs. SW Full Disk Encryption

- HW advantages:
  - Faster
  - Totally transparent, works for any OS
  - Setup probably easier
- HW disadvantages:
  - Not ubiquitously available today
  - More expensive (not that much, though)
  - Might not fit into a particular machine
  - Backward compatibility

Example of Hardware Full Disk Encryption

- Seagate's Momentus 7200 FDE line
- Hardware encryption for entire disk
  - Using AES
- Key accessed via user password, smart card, or biometric authentication
  - Authentication information stored internally on disk
  - Check performed by disk, pre-boot
- .3 Gbytes/sec maximum transfer rate (2014)
- Primarily for laptops

Example of Software Full Disk Encryption

- Microsoft BitLocker
- Doesn't encrypt quite the whole drive
  - Unencrypted partition holds bootstrap
- Uses AES for cryptography
- Key stored either in special hardware or USB drive
- Microsoft claims "single digit percentage" overhead
  - One independent study claims 12%